

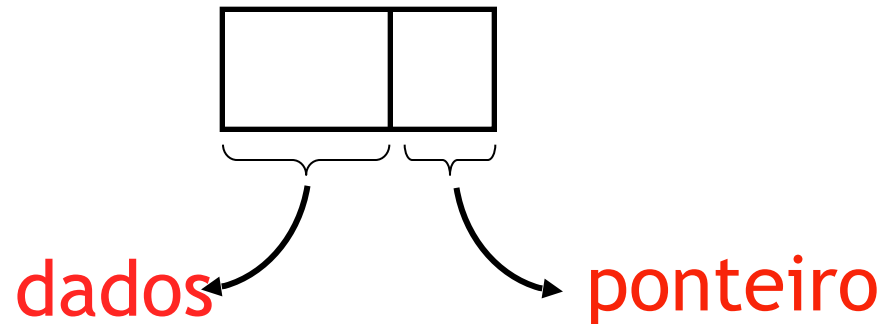
Listas Lineares

Alocação Encadeada

Alocação Encadeada

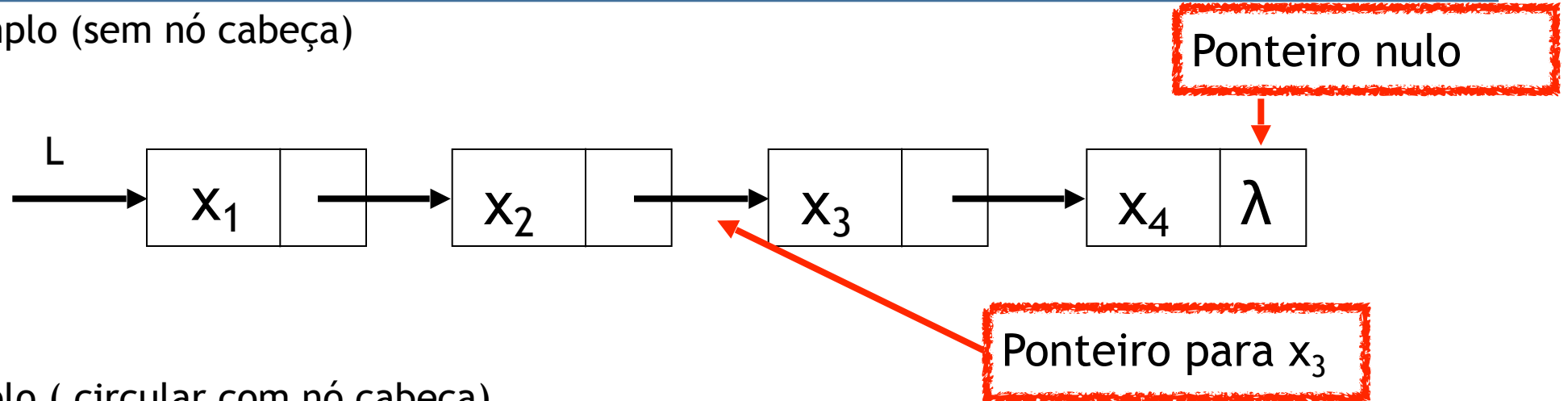
- As posições na memória são alocadas ou desalocadas na medida em que são necessárias.
- Os nós da lista encontram-se em posições quaisquer na memória e são interligados por ponteiros que indicam a posição do próximo elemento da lista.

É necessário o acréscimo de um campo em cada nó para indicar o próximo elemento.

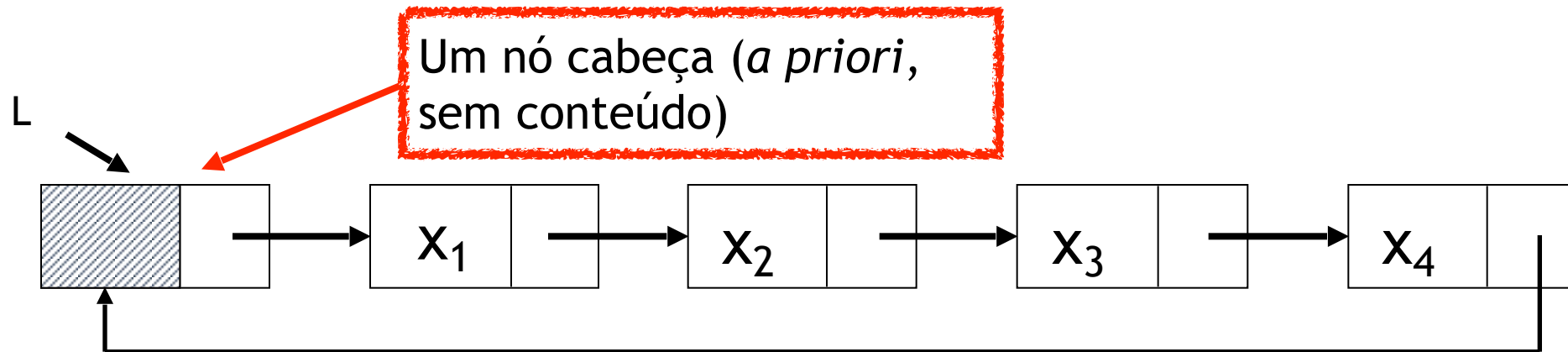


Lista Linear Simplesmente Encadeada

Exemplo (sem nó cabeça)



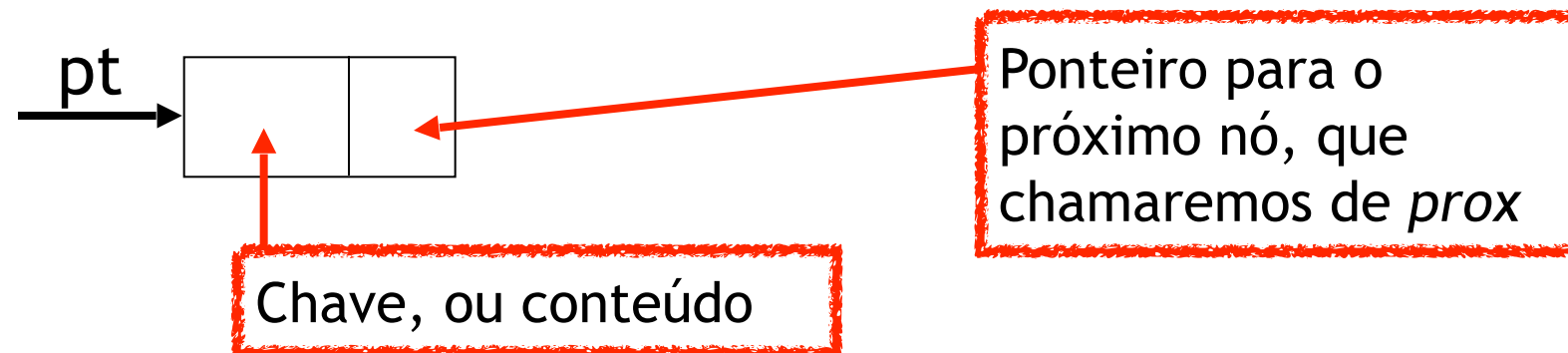
Exemplo (circular com nó cabeça)



Lista Linear com Alocação Encadeada

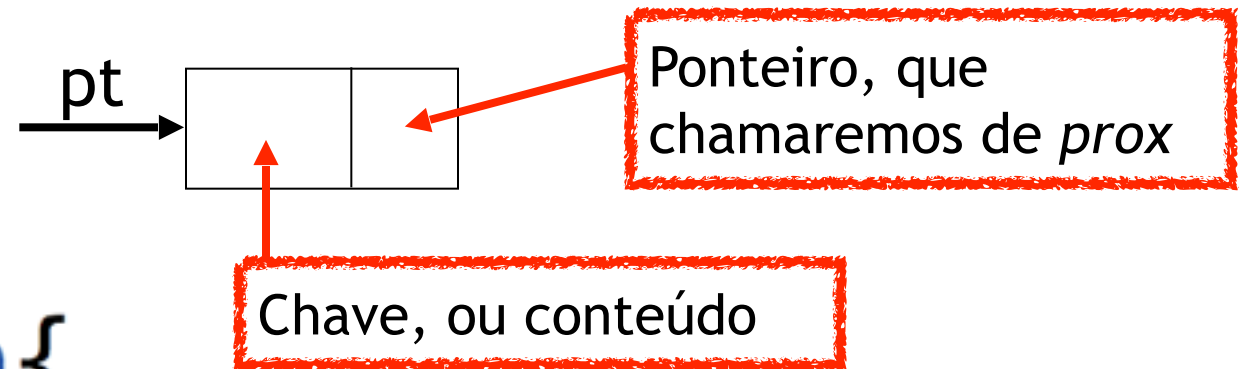
Criação e liberação de uma posição na memória

Criação de uma posição:
ocupar(pt)



Liberação de uma posição:
desocupar(pt)

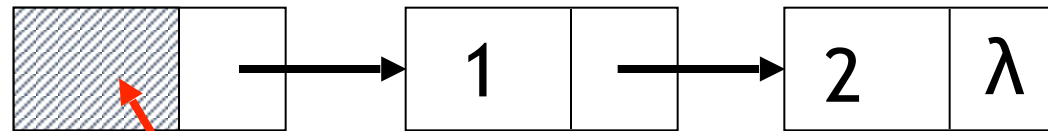
Lista Linear com Alocação Encadeada



```
typedef struct nn{  
    int chave;  
    struct nn *prox;  
} No;
```

Exemplo C++: Lista Linear com Alocação Encadeada

```
1 No *ptlista = new No();
2 No *no1 = new No();
3 No *no2 = new No();
4 ptlista->prox = no1;
5 no1->prox = no2;
6 no2->prox = NULL;
7 no1->chave = 1;
8 no2->chave = 2;
9 for (No *it = ptlista->prox; it!=NULL; it = it->prox){
10     cout<<it->chave<<endl;
11 }
```



Um nó cabeça
(ptlista)

Busca em Lista Simplesmente Encadeada Ordenada com cabeça

Busca(x :inteiro, ant :ponteiro, $pont$:ponteiro)

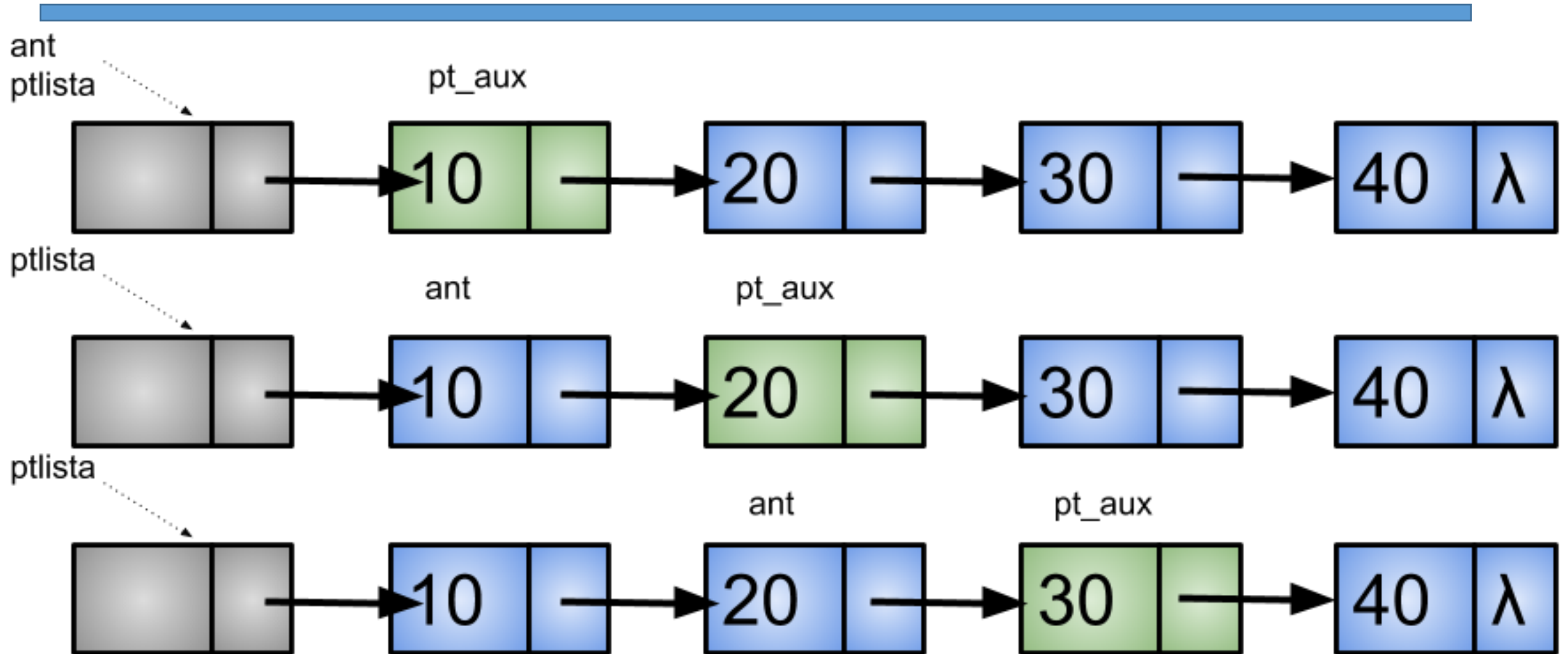
```
ant ← ptlista; pont ← λ
pt_aux ← ptlista↑.prox
enquanto(pt_aux ≠ λ) faça
    se (pt_aux↑.chave < x) então
        ant ← pt_aux
        pt_aux ← pt_aux↑.prox
    senão
        se (pt_aux↑.chave = x) então
            pont ← pt_aux
        pt_aux ← λ
```



Note que *ant* e *pont* são PONTEIROS

Complexidade?

Busca em Lista Ordenada (x=30)

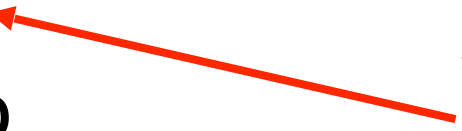


Inserção

Insere(x)

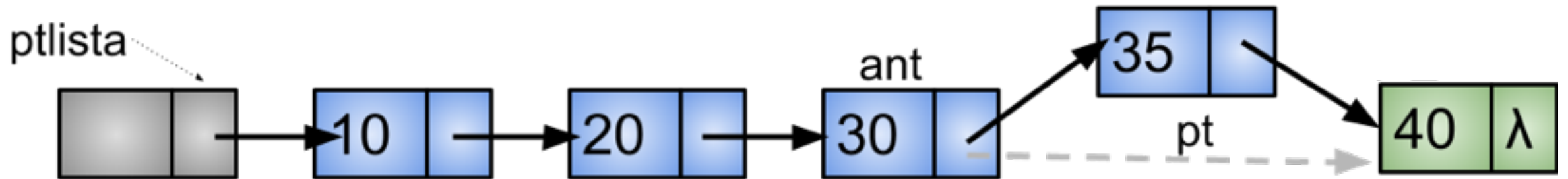
Busca(x, ant, pont)
se (pont = λ) então
 ocupar(pt)
 pt↑.chave \leftarrow x
 pt↑.prox \leftarrow ant↑.prox
 ant↑.prox \leftarrow pt
senão “Elemento já se
 encontra na lista”

Complexidade?



A Busca retorna *pont*, que pode ser λ (elemento ausente) ou pode ser o ponteiro para o nó do elemento encontrado. Ela torna também *ant*: o nó imediatamente anterior onde o elemento (deve) deveria estar.

Inserção



ocupar(pt)

$pt \uparrow .chave \leftarrow x$

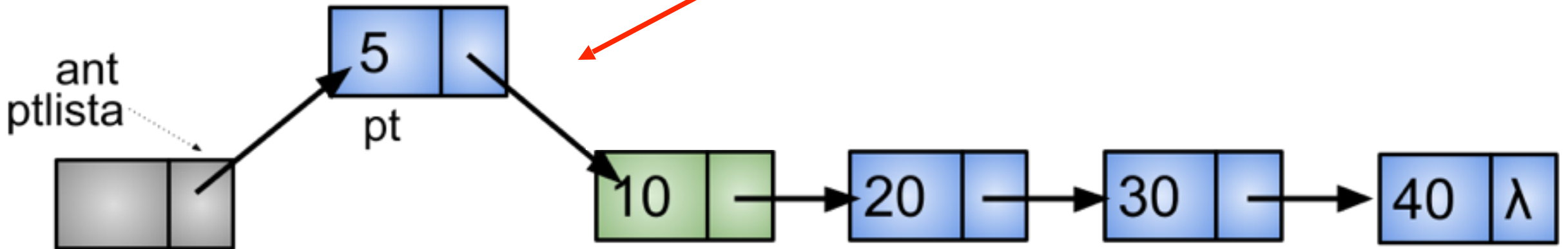
$pt \uparrow .prox \leftarrow ant \uparrow .prox$

$ant \uparrow .prox \leftarrow pt$

Inserção

```
ocupar(pt)  
pt↑.chave ← x  
pt↑.prox ← ant↑.prox  
ant↑.prox ← pt
```

Note que, aqui,
temos $ant == ptlista$.
Ou seja, o nó cabeça
($ptlista$) facilita a
inserção.



Remoção

Remove(x)

Busca(x, ant, pont)

se (pont $\neq \lambda$) então

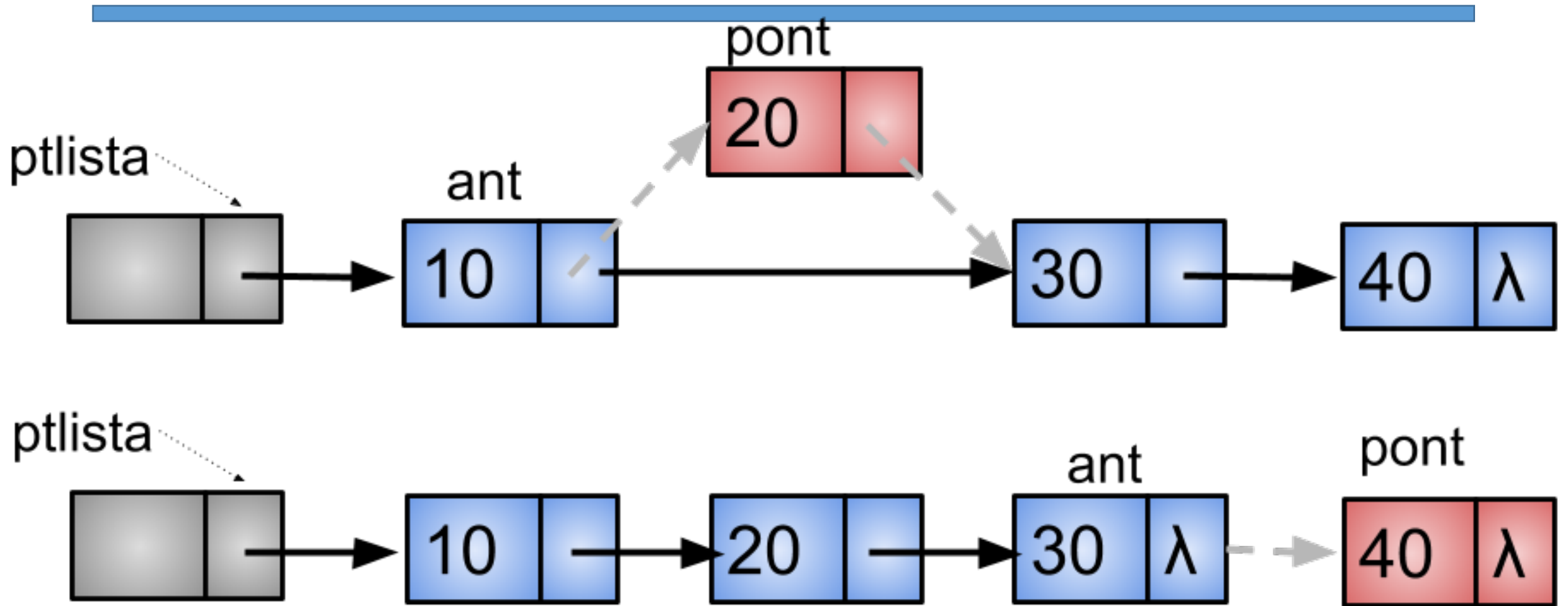
ant↑.prox \leftarrow pont↑.prox

desocupar(pont)

senão “Elemento não se encontra na lista”

Complexidade?

Remoção



Listas sequenciais vs Listas encadeadas

Listas sequenciais

- Acesso em tempo constante $O(1)$;
- Inserção e remoção precisam de deslocamento
- Precisa alocar previamente a lista inteira
- ...

Listas encadeadas

- Acesso em tempo linear
- Inserção e remoção não precisam de deslocamento
- Precisa alocar um elemento cada vez que for criado
- ...

Pilhas

empilha(x)

ocupar(pt)

pt↑.chave ← x

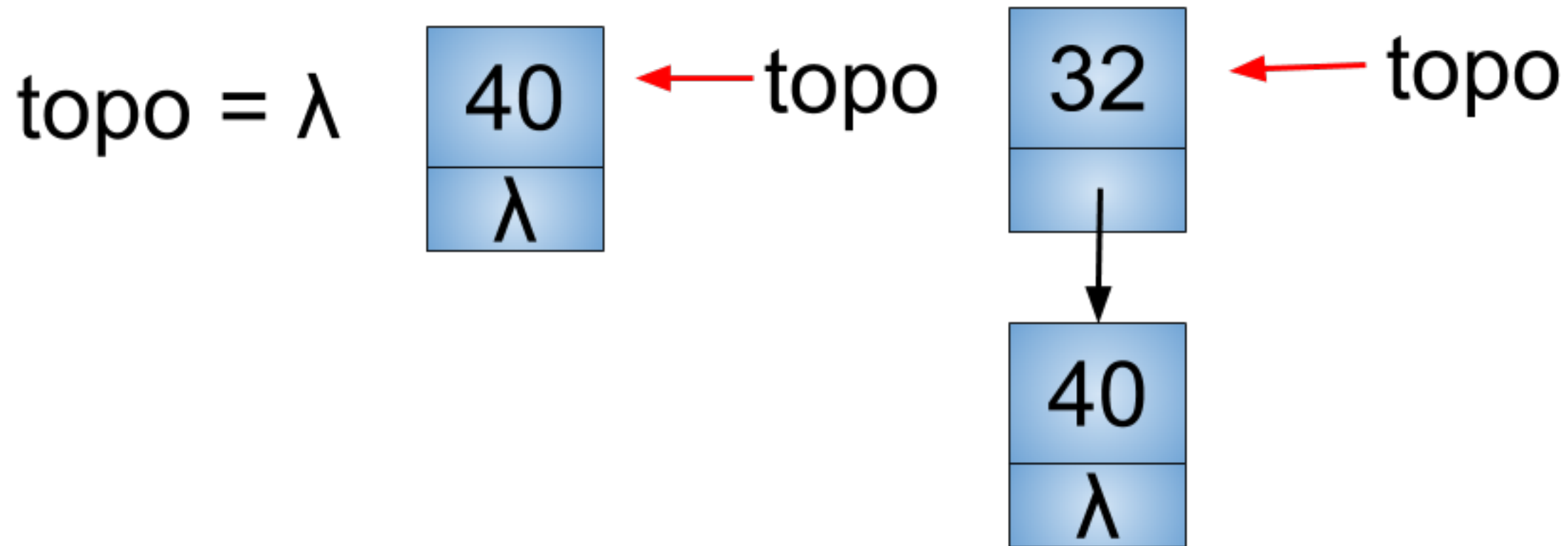
pt↑.prox ← topo

topo ← pt

Pilha vazia \rightarrow topo = λ

Complexidade?

Pilhas



Pilhas

desempilha()

se (topo \neq λ) então

pt \leftarrow topo

topo \leftarrow topo \uparrow .prox

valor \leftarrow pt \uparrow .chave

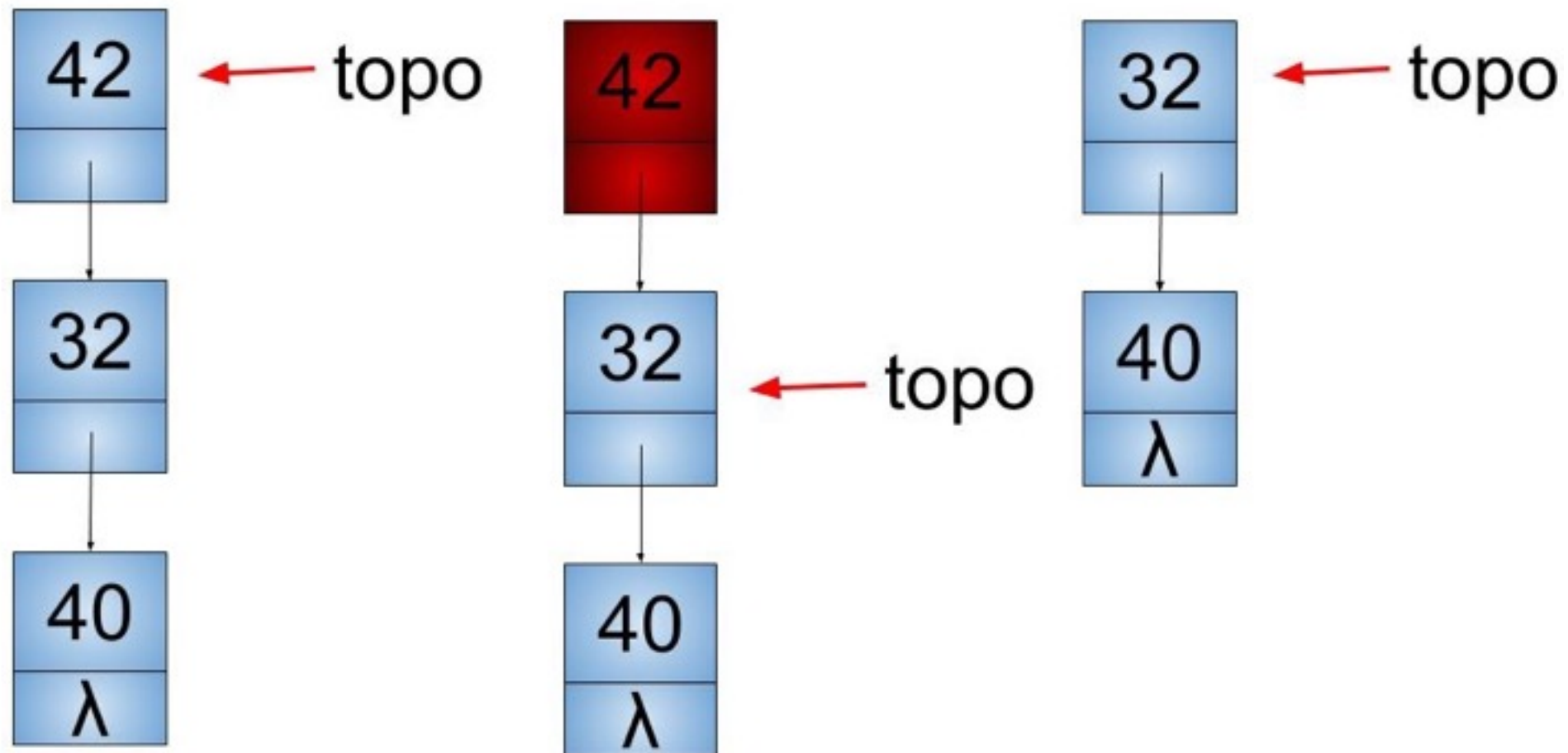
desocupar(pt)

senão “Pilha vazia”

Pilha vazia \rightarrow topo = λ

Complexidade?

Pilhas



Filas

Insere(x)

ocupar(pt)

pt↑.chave ← x

pt↑.prox ← λ

se (fim ≠ λ) então

 fim↑.prox ← pt

senão

 inicio ← pt

 fim ← pt

Dois ponteiros:

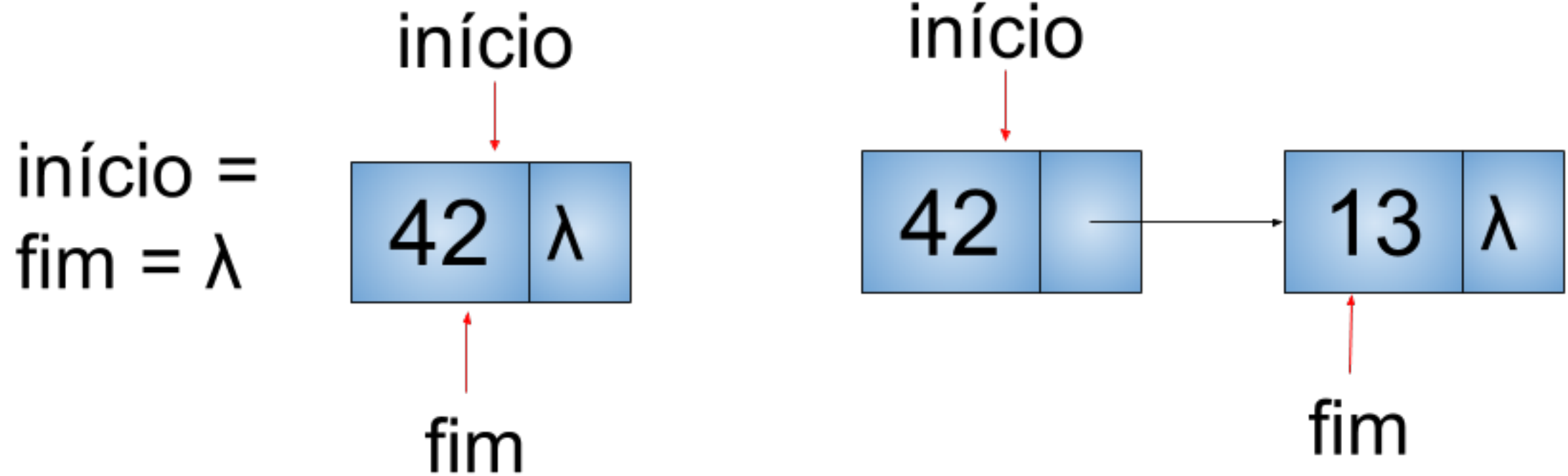
inicio => início da fila

fim => fim da fila

inicio = fim = λ => fila vazia

Complexidade?

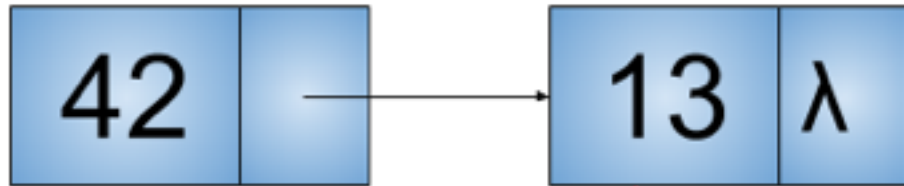
Filas



Filas



início



fim

início



fim

Filas

Remove()

se ($\text{inicio} \neq \lambda$) então

pt \leftarrow inicio

inicio \leftarrow inicio \uparrow .prox

se ($\text{inicio} = \lambda$) então

fim $\leftarrow \lambda$

valor \leftarrow pt \uparrow .chave

desocupar(pt)

senão “Fila Vazia”

Dois ponteiros:

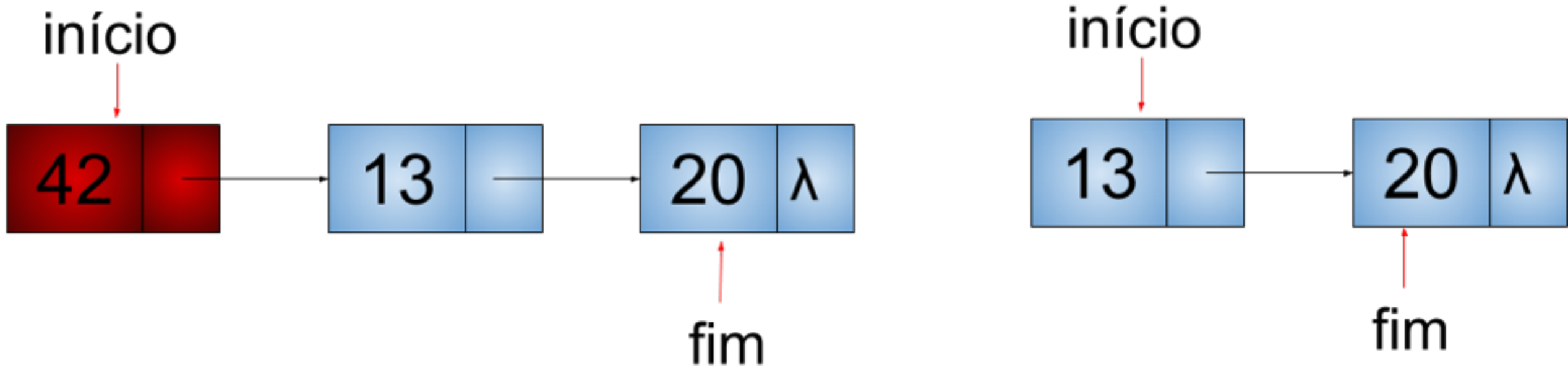
inicio \Rightarrow início da fila

fim \Rightarrow fim da fila

inicio = fim = λ \Rightarrow fila vazia

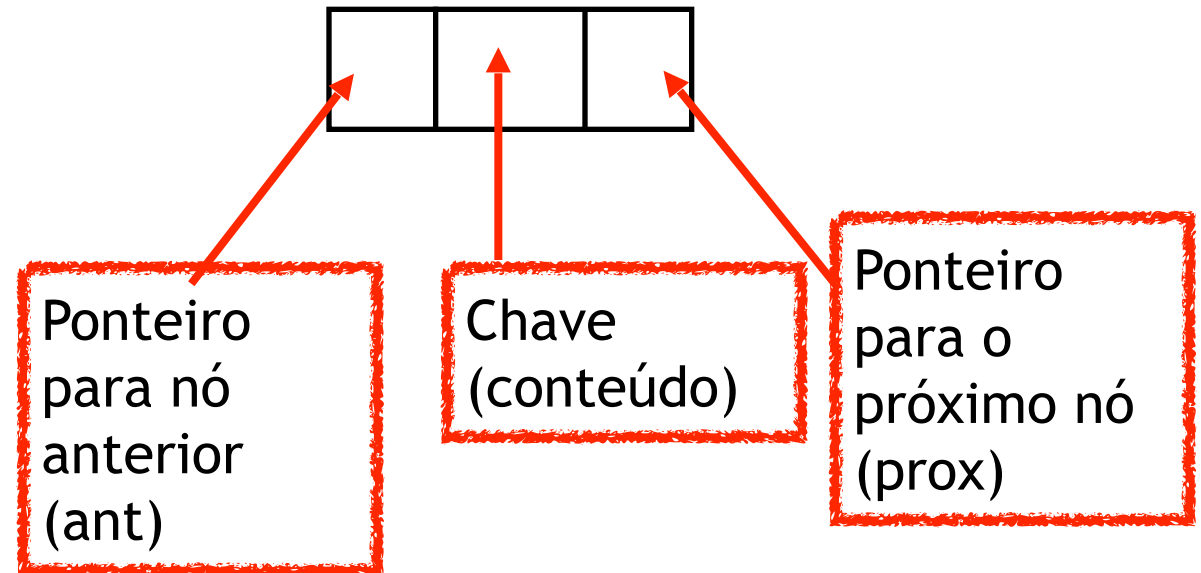
Complexidade?

Filas

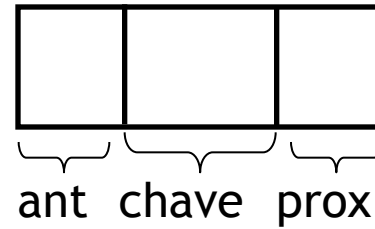


Lista Linear Duplamente Encadeada

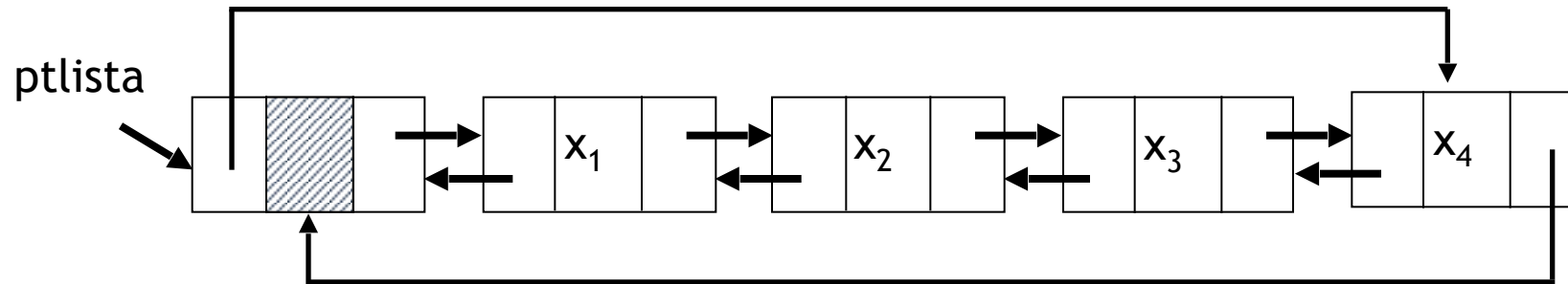
```
typedef struct nn{  
    int chave;  
    struct nn *prox;  
    struct nn *ant;  
} No;
```



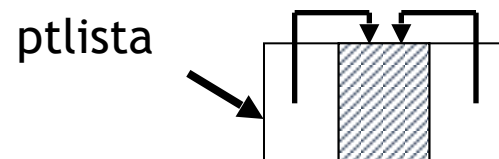
Lista Linear Duplamente Encadeada



Exemplo (circular com nó cabeça)



Lista Vazia



Busca em Lista Duplamente Encadeada Ordenada

```
Busca(x) //lista duplamente encadeada com nó cabeça
    ultimo ← ptlista↑.ant
    se x ≤ ultimo↑.chave então
        pont ← ptlista↑.prox
        enquanto (pont↑.chave < x) faça
            pont ← pont↑.prox
        retorna(pont)
    senão retorna(ptlista)
```

Complexidade?

Inserção em Lista Duplamente Encadeada

Insere(x)

 pont ← Busca(x)

 se (pont = ptlista) ou (pont↑.chave ≠ x) então

 anterior ← pont↑.ant

 ocupar(pt)

 pt↑.chave ← x

 pt↑.ant ← anterior

 pt↑.prox ← pont

 anterior↑.prox ← pt

 pont↑.ant ← pt

 senão “Elemento já se encontra na lista”

Complexidade?

Exercício

1. Fazer o algoritmo de remoção em lista duplamente encadeada



FIM